Security Policy Automation – from Specification to Device Configuration

Kirk Schloegel, Tom Markham, Walt Heimerdinger
Honeywell Laboratories, 1985 Douglas Dr, Golden Valley, MN 55422, USA
Alberto Schaeffer-Filho, Morris Sloman, Emil Lupu
Imperial College London, 180 Queen's Gate, London SW7 2AZ, UK
Seraphin B. Calo, and Jorge Lobo
IBM Research, 19 Skyline Drive, Hawthorne, NY 10530

Abstract— To achieve the potential of network centric warfare the Army must securely share information across US operational units and with coalition partners while at the same time denying our enemies access to sensitive information. The sheer number of configuration parameters necessary to achieve such secure interoperability and optimal data sharing creates the opportunity for human error and slows the deployment process. Furthermore, the underlying security policies may be dynamic due to changing missions, changing coalition partner relationships and compromise of devices. Finally, the MANET environment is often bandwidth limited, links are sometimes intermittent and end-to-end connectivity is not always possible.

***Index Terms*—security policy, natural language, automation, Ponder,**

I. Introduction

Security policy is complex, error prone and time consuming.

This impacts the development instantiation and update of policy with coalition partners. Broad challenges include:

- Deconfliction and validation require an understanding of policy interaction and device capability.
- Policy instantiation often involves imprecise or inconsistent interpretations on the path to device configuration.
- The MANET environment is dynamic and lacks much of the infrastructure typically found in wired enterprise environments.

The impact on the warfighter is that errors and delays create situations in which the data is not available to those who need it in a timely manner. Errors can also lead to the compromise of sensitive information.

This paper describes the technical approach being developed by the International Technology Alliance[1] to address these challenges. The ITA approach is to create a layered model which allows successive refinement, validation, distribution and update of policy. This approach

- Reduces complexity by allowing specification of policy using constrained natural language
- Reduces errors by applying formal methods and automated reasoning to identify conflicts, inconsistencies, ambiguity and gaps in device policy enforcement capability.
- Eliminates errors associated with human interpretation by automating the policy refinement and device configuration steps.
- Reduces delays by automating both the policy processing and distribution of policy updates and associated status.

Layered Model Overview – Figure 1 below shows the concept of the layered model. The major layers and their functions are:

- The **Policy Specification Layer** consists of constrained natural language grammars that are conducive to the specification of security policies, tools to support the authoring of syntactically-correct policies and tools and ontologies to enable the transformation of natural language polices into abstract policies.
- The **Abstract Policy Layer** automatically analyzes sets of abstract policies for semantic correctness and consistency through a number of formal methods.
- The **Concrete Policy Layer** automatically transforms correct and consistent abstract policy sets into concrete policy sets that must be upheld by the different components of the distributed system to meet the policy goals.
- The **Executable Policy Layer** transforms and distributes concrete policy sets to specific devices before and during

[1] Research was sponsored by US Army Research laboratory and the UK Ministry of Defence and was accomplished under Agreement Number W911NF-06-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the US Army Research Laboratory, the U.S. Government, the UK Ministry of Defense, or the UK Government. The US and UK Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
01 DEC 2008	**N/A**	**-**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Security Policy Automation from Specification to Device Configuration	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Honeywell Laboratories, 1985 Douglas Dr, Golden Valley, MN 55422, USA**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release, distribution unlimited

13. SUPPLEMENTARY NOTES
See also ADM002187. Proceedings of the Army Science Conference (26th) Held in Orlando, Florida on 1-4 December 2008, The original document contains color images.

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**	**UU**	**8**	

deployment and provides status reporting. The policy infrastructure at this layer determines when policy conditions are met. This layer also reports device discovery information back up to the Concrete Policy Layer.

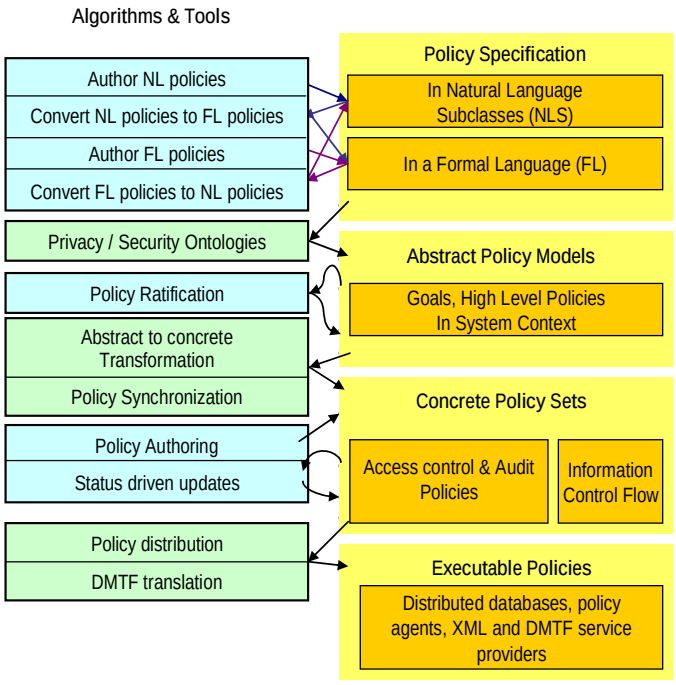

Figure 1 The four layers and functions within the policy model

The value to the warfighter is that these layers are being integrated so that instead of islands of research there will be a top to bottom integrated policy system. Each of the layers is discussed below.

II. NATURAL LANGUAGE

There are a number of methods that might be used to enable usable policy creation. We have adopted one approach using constrained natural language.

Figure 2 Policy Specification, Editing & Analysis provides an overview of the architecture for a generalized policy creation utility. Such a utility is focused on creating the technical capabilities for organizations to specify understandable security policies, and link the authored policies with their implementations across their IT configuration.

Research has shown that organizational policies are authored by individuals with a range of skills. Some policy authors have a legal and/or business background while others have more technical backgrounds. In order to support users with a variety of skills, the authoring tool has been designed with two methods for specifying policies.

Policy authors can write policy rules in natural language using a rule guide or they can import existing text policies and tailor them using the rule guide. The tool then transforms the natural language into a structured format. Alternatively, policy authors can use a structured format directly to define the elements and rule relationships. The tool will generate the corresponding natural language versions for rules created using this method. Authors can use either method exclusively or move between the two methods and the tool will keep the two formats synchronized. Once the policy is in the structured format, visualizations of the policy are provided to assist the policy creators in ensuring that the policy coverage is what was intended. Also, analysis capabilities are provided to identify conflicts and redundancies among policy rules within a policy and between policies. Finally, when the policy author is satisfied with the policy, the tool generates the policy rule in the desired format (e.g., XACML, ACPL, CIM-SPL, Ponder), based on the structured natural language. The following paragraphs provide some details about the use of the tool for authoring and viewing policies.

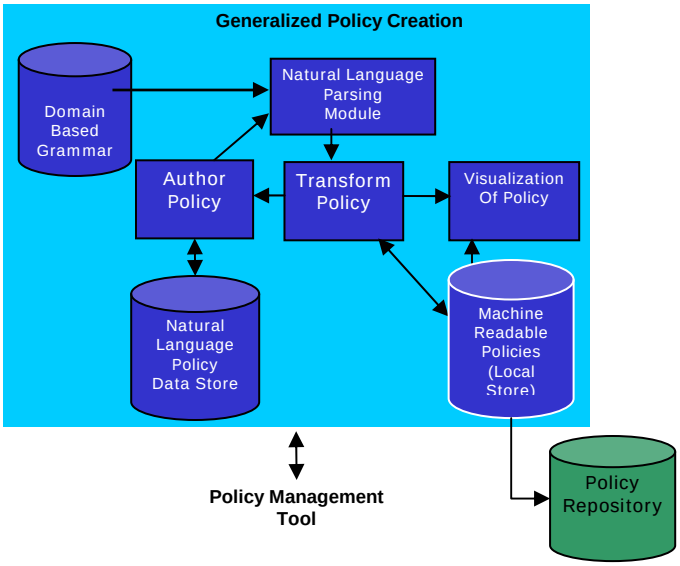

Figure 2 Policy Specification, Editing & Analysis

When a user logs onto the authoring tool, the Policy Selection page is displayed. Here the author can select an existing policy to modify, upload an existing text file, or create a new policy file. The user enters a policy name and selects the policy domain. Once the policy is selected the author will be taken to the natural language policy authoring page. The area at the top of the page shows the policy name, description, domain, and the date it was first created and last modified. The rule guide that is shown above the policy text editing area has two purposes. First it reminds authors of what elements are needed in an implementable policy rule. We define an implementable policy rule as a rule that can be defined for automated enforcement through technology. Second, the guide defines the order in which elements in a policy rule must be placed so that the natural language parsing technology can identify them with as high a degree of accuracy as possible. Security policies have been found to be structured in the following manner: **[User Category(ies)]** can perform **[Action(s)]** on **[Data Category(ies)]** for **[Purpose(s)]** if **[(optional) Condition(s)]** with **[(optional) Obligations].**

The author can edit text in the policy text editing area or can cut and paste text from other files so long as the rules conform to the rule guide. However, the author does not have to use the exact wordings shown in the guide. It simply increases the accuracy of the parsing to do so. When parsing has completed, the user proceeds to the Structured Policy Authoring Method page to see the structured format of the policy.

When a policy rule is parsed, its elements are saved (i.e., user categories, actions, data categories, purposes, conditions and obligations). The elements are reconstructed into sentences and shown next to radio buttons in a list. While the accuracy of the parser is very high, it is not perfect so it is important for the user to compare the text of the parsed rules with the original rule text. Additionally, all of the rule elements in the selected rule are shown in rule element lists that appear below the reconstructed rules on the page. Rule element lists are: initially populated with domain defaults (e.g., typical terms for the domain of the policies); extended as new elements are found in natural language rules during parsing; and, explicitly added to by the author using input fields provided.

If the user wishes to change a rule, it can be selected by clicking on the radio button next to it, then the elements of the rule can be selected from the policy element lists, and the changes will take effect when the "Modify Rule" button is clicked. A single rule or all the rules in the policy can be created using this method. If the user returns to the authoring page, all the changes that have been made on the structured policy authoring method page will be reflected in the text. When the policy author is satisfied with the policy they can generate an XACML version of it by clicking on the "Save as XACML" (or "Save as ACPL", etc.) button at the bottom of the page.

One of the unique capabilities of the tool is its use of natural language parsing. To provide this functionality the tool employs a shallow parser to identify the expected elements of the policy. Shallow parsing identifies linguistic structures in natural language, but does not identify the semantic meaning of the text. The shallow parser used in the tool iteratively processes text in a number of stages by inserting meta-tags into the text to identify first parts of speech and then more application specific text items. The beginning stages operate with limited linguistic knowledge to identify syntactic structures such as nouns, noun phrases, verbs, verb groups, and modifying phrases. Using these first tags, the shallow parser then uses one or more grammars to identify the desired text in a document based on patterns of parts of speech. As part of this research, grammars were designed to identify five policy element types in the policy rules. These include user categories / roles, actions, data categories, purposes, and condition / obligations. These element types were chosen because they define the elements that are specified by security policies.

The analysis of natural language is a very difficult task even in a limited domain. In order to employ natural language with a high enough degree of accuracy to make the functionality useful and usable, the tool was designed to analyze constrained natural language rather than full natural language. Two constraints are placed on the policy rules. First, each policy rule must be contained in a single sentence. This allows the parser to easily identify the beginning and end of each rule. Second, policy rule elements are expected to be in one of two possible orderings. The first ordering follows the guide, and the second ordering has the user category and data category reversed to support a passive voice form of the rules. Improving the accuracy and generality of the grammars is an ongoing research effort.

Policy rules can be checked for correctness at a variety of levels. For example, a single policy rule can be marked as in error if the authoring component detects a syntax error. It is possible to carry out the analysis on high level policies (e.g., to detect on the parsed syntactic level in natural language policies that a purpose is missing), or on policies refined into formal representations which include semantic detail for the objects and actions in the policy rule (e.g., to detect that an object in an XACML policy is not of the proper type or class).

In a similar fashion, policy sets can be analyzed to determine whether the policy rules in the set are in conflict in a number of ways. At the syntactic level, we can carry out limited checks for redundancies and conflicts. At more formal levels of representation, the semantic information associated with the policy rule can be used to carry out additional analysis, such as examining policy coverage over a space of possible values or investigating conflict at a more detailed level of analysis. We believe that there are a wide range of analysis algorithms that might be developed and implemented for security policies, and that these can be made useful for policy authors.

Architecturally, we believe that it is useful to think of analysis as potentially applying to policies at different levels of abstraction (i.e., not just to high or low level policies). Thus, in a policy management framework which includes authoring and refinement through implementation, policy analysis can occur at any level. Furthermore, our experience suggests that analysis at the high level of abstraction is useful even if it is not complete. That is, even though we can not detect all conflicts that might be in a policy by analyzing just syntactic elements, it can be valuable to authors to indicate those problems we can find as early as possible, before a great deal of time has been invested in mapping policy elements to configuration objects.

At the specification level, if policies are written in natural language, there must exist a mapping from terms in the policy specification to structures in the abstract model. For example, if a policy refers to encryption mechanisms, the formal model must be able to determine what encryption is and what encryption mechanisms exist in order to be able to rule out as invalid a policy that tries to enforce an encryption mechanism that is not supported by the concrete layers below. Validation continues to happen at the lower layers because policies get transformed and can possibly be split into multiple policies that may be enforced by different end devices or elements. Some mechanisms may be supported by some but not all end devices, and from the specification, it might be difficult to detect how the policy will be transformed and what devices

will be affected to be able to decide if the policy will be valid at the execution level.

III. ABSTRACT POLICY

Natural language policies are converted to an abstract language for analysis. To be able to capture both the dynamic aspect of the system where policies are enforced and the policies themselves we have used logic based action description languages for planning as the basis for policy description at the abstract layer. The main two advantages of such a language are first that we can use logic programming to provide formal semantics to policies – a requirement for analysis, and second that we can use all the tools develop for logic programs to study and analyze policies.

Space limitations prevent us to present the full language. We will present a couple of examples so that reader can get a general overview of the language. Let us start with following example:

Alice can delete classified data files from her device if she sends a notification to the supplier of the data 10 minutes in advance

There are two operations or actions mentioned in the policy: notify and delete. We appeal to the reader's intuition for the attributes of the terms and operations that will be used through the examples. The subject of the authorization is alice. The target is alice's device where the file resides. To specify authorization policies we will make use of the following domain independent terms:

1) *req(Subject, Target, Action, Time)*
2) *do(Subject, Target, Action, Time)*
3) *deny(Subject, Target, Action, Time)*
4) *permitted(Subject, Target, Action, Time)*
5) *denied(Subject, Target, Action, Time)*

req is an input term that will come as a request from the environment, *do* and *deny* will be the responses of the policy systems to requests, *permitted* and *denied* will be defined by the policies. Intuitively the time argument in all the terms can be interpreted as the point in the execution where the term is being evaluated. The example makes use of another group of subjects: Suppliers of data. The following is a domain dependent predicate needed for the specification:

filedesc(Supplier,Name, Type, Time)

The policy can be (partially) described by the following rules:

*do(alice, S, notify(delete(F)), T0) & filedesc(S, F, class, T0) & **not** reqInBetween(S, F, retain(F), T0, T1) & T1 = T0 + 10mins → permitted(alice, device, delete(F), T1)*

req(Sub, Tar, Act, Tm) & Tl ≤ Tm ≤ T → reqInBtween(Sub, Tar, Act, Tl, T)

The second rule check for request between the time interval *[Tl,T]*. For availability we need to make sure that if the request to execute an action appears in the trace of a system and the subject making the request is permitted to execute the action the action is executed. We achieve that by adding the following domain independent rule to our policies:

req(Sub, Tar, Act, T) & permitted(Sub, Tar, Act, T) → do(Sub, Tar, Act, T)

Similar rules can be written to describe *denied* policies. This type of policies is called (positive/negative) authorization policies. In addition the abstract language also has obligations, e.g. policies that impose the obligation to execute an operation to some entity in the system. An example of obligation is:

A connecting node must provide a second identification within 5 minutes of establishing a connection

Standard components of an obligation are: the Subject being acquiring the obligation (e.g. the connecting node); target of the action of the obligation (e.g. the device where the node is connecting to); the action of the obligation (e.g. provide second id); and event that triggers the obligation (e.g. establishing the connection). The terms used in obligations are:

1) *obl(Subject, Target, Action, T1, T2, Time)*
2) *fulfilled(Subject, Target, Action, Time)*
3) *violated(Subject, Target, Action, Time)*

The obligation can be encoded with the rule:

node(U, T) & do(U, server, connect(U, server), T) → obl(U, server, submit2ID(U, server), T, T + 5min, T)

The rules for *fulfilled* and *violated* are:

obl(Subject, Target, Action, T1, T2, T) & do(Subject, Target, Action, T) & T1 ≤ T ≤ T2 → fulfilled(Subject, Target,Action, T)

obl(Subject, Target,Action, T1, T2, T) & T2 < T → violated(Subject, Target,Action, T)

The abstract language for policies is sufficiently expressive that many different formalisms (e.g. Ponder2 [RDD07], XACML [OAS05], Cassandra [BS04,BN07]) can automatically be translated into it. Automated translation algorithms have been developed for a large class of Ponder2 described in the next section.

We are able to do application dependent and independent analysis of policies such as:

– Modality conflicts such as the acquisition of an obligation without the permissions necessary for its fulfillment.
– Separation of duty clashes, including static separation of duty, dynamic, and many other classes.
– Coverage gaps, where no policy exists to dictate what the correct response to a request should be.
– Policy comparison, including the question of whether two policies are equivalent or one is contained in the other.
– Behavioral simulation, where specific sequences of requests and events in the policy-regulated system are entered, to see the policy decisions which arise during the run.

We use abductive, constraint logic programming (ACLP) systems as the basis of our analysis algorithms and

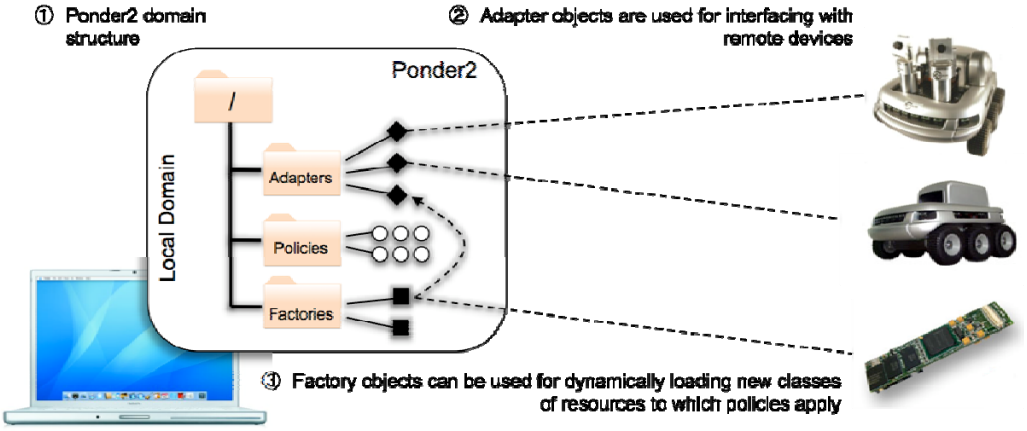

Figure 3 - Ponder2 domain containing management policies, adapters and factory objects

implementation, and the Event Calculus (EC) [KS86] to describe how events and actions occurring in the system affect the system states. The EC allows us to specify how actions and events change system state, leading to circumstances in which a given policy rule is applicable. This information is an output of the analysis. The reader can find details of the analysis and the language in [CL08].

IV. CONCRETE POLICY

The output of the abstract language analysis is transformed to a concrete policy language with an associated framework for representing physical devices. This section presents an overview of the Ponder2[2] policy framework, and the *PonderTalk* language used to specify policies in the concrete layer and to interact with Ponder2 interpreters. In addition to the types of policies supported and the overall functioning of Ponder2, this section focuses on two distinguishing aspects of the framework, which are highly relevant to a MANET environment: *(a)* the ability to support management policies written for an extensible array of resources (through *adapter objects*), and *(b)* the flexibility to load on demand all the code needed by these policies (through *factory objects*).

A. Types of Policies

Policies are written in a high-level language called *PonderTalk*, more generally used to control and to send commands to the Ponder2 interpreter. Ponder2 caters for two types of policies: *obligations* and *authorisations*.

Obligation policies are *event-condition-action (ECA)* rules that define the management actions that must be performed in response to events, provided a set of conditions is fulfilled. These policies cater for the adaptative behaviour of resources. Obligations are of the form:

on *<event>* **do**
if *<conditions>* **then**
<target> <action>

To simplify notation an obligation policy can have a list of target-action pairs, all evaluated when the event occurs and the condition holds true. Implicitly, the Ponder2 interpreter enforcing this policy is called the *subject* of the obligation, and the action is invoked on a *target* resource.

Authorisation policies specify what actions a *subject* is allowed (positive authorisation) or forbidden (negative authorisation) to invoke on a *target* resource or service. Authorisations are of the form:

auth[+/-] <subject> → **if** <condition> **then**
<target> <action>

While obligations are enforced by the (implicit) subject of the policy, authorisation decisions are typically enforced by the target of the policy, as we assume target resources protect themselves against unauthorised access.

Ponder2 policies are interpreted, meaning they can be easily changed without shutting down or recoding components. These policies are instantiated by the local Ponder2 interpreter, or loaded remotely in the form of *missions* from other Ponder2 instances [LD08].

B. Adapter Objects

In order to provide a uniform interface for policy specification, Ponder2 maintains *adapter objects* for each of the components on which management actions can be performed. Adapter objects abstract the specific communication protocol and the interface used when interacting with heterogeneous resources. We have used adapter objects to enact management policies on remote BSN nodes using IEEE 802.15.4 radio, and on remote services using Java RMI and SOAP, for example. Adapter objects to remote resources may be held transparently in a Ponder2

[2] http://www.ponder2.net

instance, thereby facilitating the deployment of management policies across distributed resources.

Adapter objects (also called *managed objects*) are grouped in a domain structure that implements a hierarchical namespace, where managed objects are addressed using path expressions. Policies may specify management actions to be executed on individual objects, or on entire domains, in which case the policies will apply to all managed objects inside that particular domain.

Policies are also treated as managed objects on which actions can be performed. Thus events may trigger obligation policies that perform actions on other policies (e.g. an obligation policy triggered by a *"code red"* event may disable a subset of the current security policies, while enabling a replacement group of policies). This mechanism allows policies to be dynamically added, removed, enabled and disabled to change the behaviour of the Ponder2 instance (or the device running it) without interrupting its functioning.

C. *Factory Objects*

Ponder2 has the ability to load all the code needed on demand through the use of *factory objects*. Factories provide a high degree of flexibility to Ponder2, in that adapter objects for remote devices may be dynamically created or even new types of policies can be defined (e.g. *delegation*, *filtering*, etc) by providing and dynamically loading the corresponding factory.

This makes Ponder2 suitable for a wide variety of applications and devices with different capabilities, as only those factories that are necessary in each device need to be loaded. This is particularly important when deploying Ponder2 in constrained resources, typical of MANETs, such as small portable devices carried by foot soldiers, unmanned vehicles or robots in general (a description of our experiments in using Ponder2 in this type of resources can be found in [FL08]).

When started, Ponder2 has a reference to its *root* domain only and simply recognises the *import* command, which is used to load new classes. Typically, the classes loaded are *factories* that permit the creation of new objects in domains. Factory objects are thus used to create policies and adapters for the various resources and devices to which the policies apply, thereby allowing the policy interpreter to communicate with such resources.

The overall architecture of the Ponder2 framework is shown in Figure 3. The domain structure of a Ponder2 instance contains management policies and organizes adapter objects to which the policies apply. Such adapter objects are created via factory objects, which can be loaded dynamically into the domain.

V. EXECUTABLE POLICY

The executable layer addresses the refinement of concrete policy into a common information model supporting a very wide ranges of devices. This allows the function of the adapter objects to be scaled up. The four key tasks required of the executable layer are:

1. Receive Ponder policy from the concrete layer and distribute it to policy agents within the MANET.
2. Translate the policy from Ponder to the Distributed Management task Force (DMTF) Common Information Model (CIM) as an intermediate device independent specification of security mechanisms
3. Interpret the CIM to create device specific configuration data and automatically configure the associated devices.
4. Communicate device and policy implementation status back up to the concrete layer to provide the concrete layer policy subsystem with situational awareness.

The major components within this layer are shown in Figure 4.

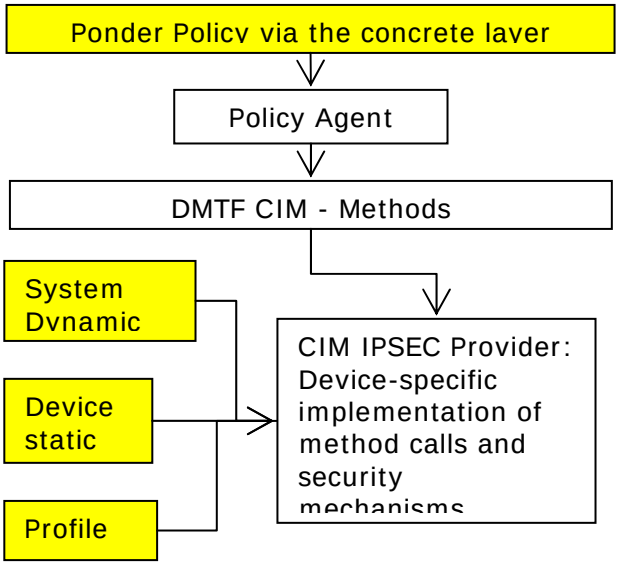

Figure 4 Major components of the executable layer

The yellow boxes highlight the 4 sources of security information used to configure the device.

- **Device static**: These are properties specific to the device itself which cannot be changed by policy. For example, the protocols (SSL, IPSEC) and cryptographic algorithms implemented in a device are device static data which influence the policies the device can implement.
- **Profile**: The profile represents security configuration information the organization has determined prior to deployment. The line between policy and the profile is fuzzy. However, the profile typically includes relatively static interoperability related data such as the asymmetric key parameters which will be used or the minimum key lengths which are acceptable. This information is loaded into the device prior to deployment.

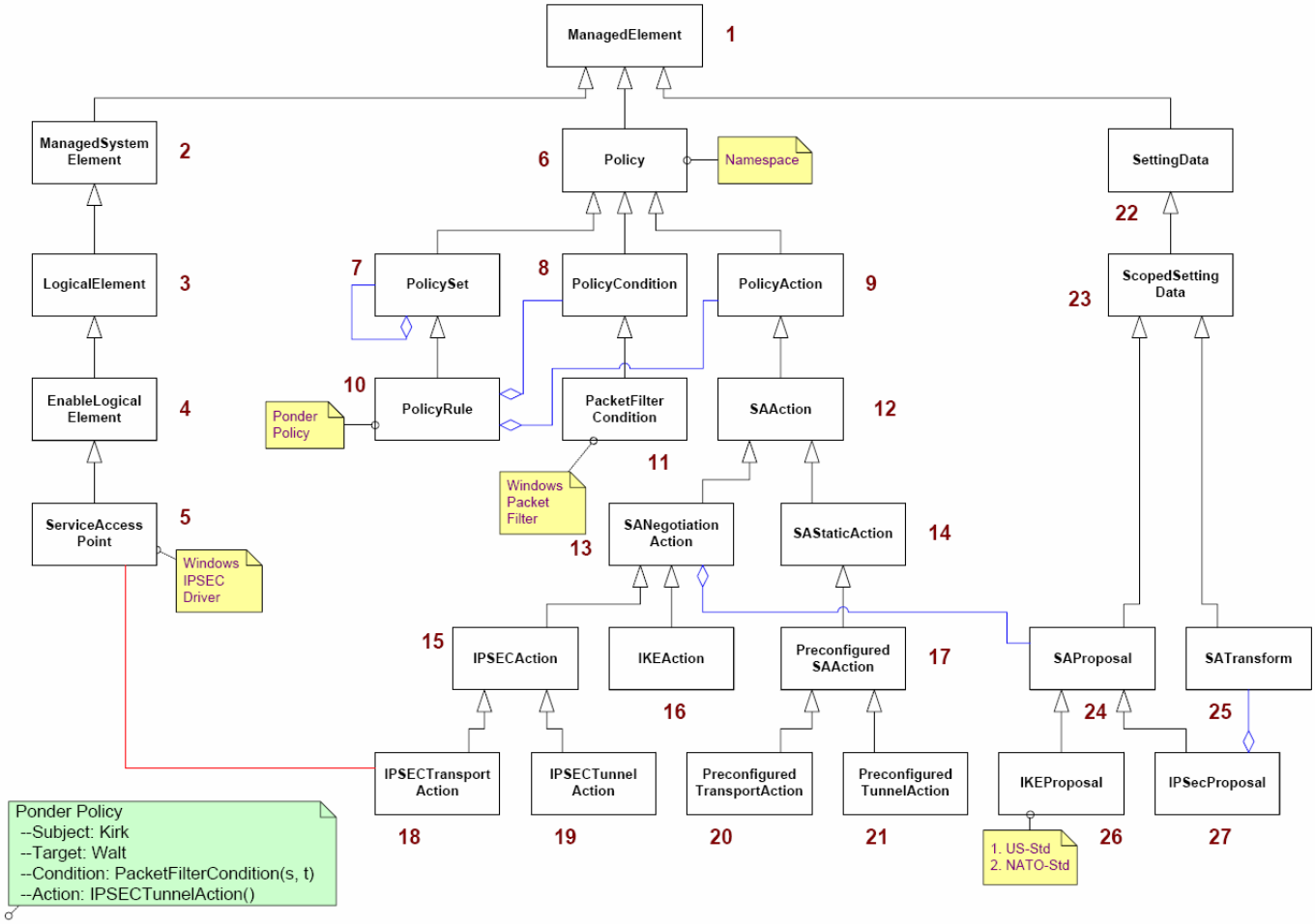

Figure 5 Common information model for IPSEC

- **Ponder policy**: This is the dynamic policy which is pushed down through the policy model and passed to the executable policy agent via the concrete to executable layer interface. This policy may change multiple times during a deployment. The policy provides information to complement the device static and profile information in order to create the device configuration.
- **System dynamic**: This represents information generated dynamically by the device itself during operation or derived from the environment. Examples include the IP address assigned to a device by a DHCP server or the IPSEC Security Parameters Index (SPI) generated by a device and associated with a security association.

An example shows how these pieces of information are combined. Assume that the organization has two devices. The **device static** properties indicate that the devices are IPSEC capable. The **profile** loaded into the device contains data values determined by the organization such as the minimum lifetime of a security association and the elliptic curve group to be used during Internet Key Exchange negotiations. The **Ponder policy** then provides information obligating device x to use IPSEC to communicate with device y. Finally the **system dynamic** information provides the locally generated security parameters index for the security association between device x and y.

One challenge is creating a mapping from the concrete policy specified by Ponder to the DMTF Common Information Model client parameters in order to provide complete, unambiguous policy specification. The mapping between the Ponder tools and the DMTF CIM for IPSEC, shown in Figure 5, highlights the need for more complete policy specification. A key research challenge is to develop a general framework which supports automated mapping of any policy specified in Ponder to the security mechanisms as represented in the DMTF CIM.

Looking at this from the device perspective the concrete policy should be able to specify any reasonable policy which can be modeled in the DMTF CIM. The DMTF CIM currently provides abstractions of common security mechanisms and services. This work must be extended to address the services and mechanisms unique to a military environment (e.g., low probability of intercept/TRANSEC keying).

Each of these abstract mechanisms and services will then be mapped upward in the layered policy model to identify the data (events, actions, targets and conditions) which must be specified implicitly or explicitly. Ambiguities and

assumptions within the policy model can lead to inconsistent or insecure policy transformations.

Once the data has been identified it will be associated with the appropriate function in the layered model (e.g., where within the policy model is the public key infrastructure fully specified to provide security and interoperability).

When dealing with security, including security policy, the devil is in the details. As the DMTF CIM is mapped upward into the policy model we expect to identify the need for additional specifications to ensure completeness as well as new algorithms to perform validation and deconfliction at multiple layers.

VI. BENEFIT TO THE WARFIGHTER

The benefits of these technologies to a warfighter operating within a MANET are:

— Rapid deployment and reconfiguration is achieved by eliminating the time required to manually configure equipment and confirm interoperability;

— Mistakes by humans translating policy to machine configuration are eliminated;

— The distribution of policy is robust against poor network connectivity because the policy is validated first then distributed via a system of delay tolerant policy agents; and,

— The advanced policy specification, distribution and implementation tools reduce the need for highly trained network administrators on the battlefield.

VII. CONCLUSION

The objectives for the ITA security policy work are to:

- Specify coalition policy within the context of a layered policy model
- Automatically transform platform-independent policies into platform-specific configurations
- Perform real time analysis and resolution of conflicts involving aggregation and composition of policies
- Develop protocols for automated policy negotiation
- Do all of the above in a dynamic, power constrained, bandwidth limited, mobile wireless network.

The approach described here addresses the challenges by breaking the policy refinement problem into functional layers which allow independent technology development within each layer. Preliminary demonstrations within the 4 layers have been performed. Future work includes:

- Creating a complete top to bottom demonstration of all 4 layers interoperating.
- Expanding the range of policies which can be processed from top to bottom.
- Enhancing the technology within each layer to address limitations specific to each layer.

VIII. REFERENCES

[LD08] E. Lupu, N. Dulay, M. Sloman, J. Sventek, S. Heeps, S. Strowes, K. Twidle, S.-L. Keoh, A. Schaeffer-Filho. *"AMUSE: autonomic management of ubiquitous e-Health systems"*, Concurrency and Computation: Practice and Experience, vol 20, issue 3. John Wiley & Sons, Ltd., 2008, pp. 277-295.

[FL08] A. Schaeffer-Filho, E. Lupu, M. Sloman, S. L. Keoh, J. Lobo, S. Calo, *"A Role-Based Infrastructure for the Management of Dynamic Communities"*, in Proceedings of the 2nd International Conference on Autonomous Infrastructure, Management and Security (AIMS). LNCS, vol. 5127. Springer, 2008, pp. 1-14.

[RDD07] Giovanni Rusello, Changyu Dong, and Naranker Dulay. Authorisation and conflict resolution for hierarchical domains. In Proc. of Int. Workshop on Policies for Distributed Systems and Networks, June 2007.

[OAS05] OASIS XACML TC. extensible access control markup language (XACML) v2.0, 2005.

[BS04] Moritz Y. Becker and Peter Sewell. Cassandra: Flexible trust management, applied to electronic health records. In CSFW, pages 139–154. IEEE Computer Society, 2004.

[BN07] Moritz Y. Becker and Sebastian Nanz. A logic for state-modifying authorization policies. In Joachim Biskup and Javier Lopez, editors, ESORICS, volume 4734 of Lecture Notes in Computer Science, pages 203–218. Springer, 2007.

[KS86] R.A. Kowalski and M.J. Sergot. A logic-based calculus of events. New Generation Computing, 4:67–95, 1986.

[CL08] Craven, R., Lobo, J., Lupu, E., Ma, J., Russo, A., Sloman, M., Bandara, A.: A formal framework for policy analysis. Technical Report, Department of Computing, Imperial College London (2008)